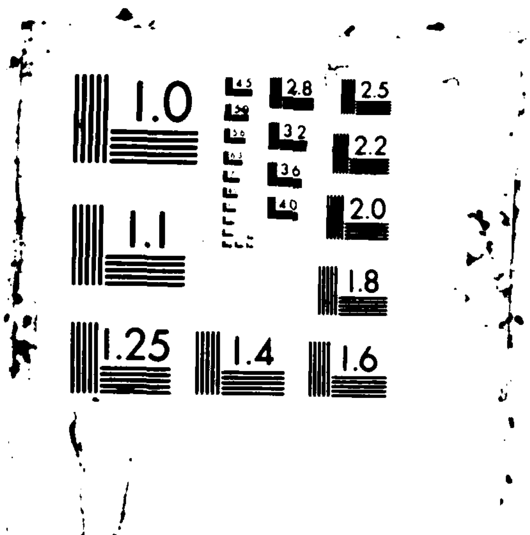
AD-A178 257   IMPLEMENTING AN INTERIOR POINT METHOD IN A MATHEMATICAL   1/1
PROGRAMMING SYSTEM I(U) KETRON MANAGEMENT SCIENCE INC
ARLINGTON VA J A TOMLIN ET AL. OCT 86 N00014-85-C-0338

UNCLASSIFIED   F/G 9/2   NL

END
4 87
DTIC



CROCOPY RESOLUTION TEST CHART

IMPLEMENTING AN INTERIOR POINT METHOD IN A MATHEMATICAL PROGRAMMING SYSTEM, I

by

J.A. Tomlin
Ketron Management Science,Inc., Mountain View, CA 94040

and

J.S. Welch
Ketron Management Science,Inc., Arlington, VA 22209

Abstract

This paper considers the integration of an interior point algorithm with a large-scale commercial MPS. Exploitation of existing features of the MPS and transition to an optimum basic solution are discussed. Preliminary computational results are presented.

Presented at the ORSA/TIMS 22nd Joint National Meeting
Miami, Florida
October, 1986

IMPLEMENTING AN INTERIOR POINT METHOD IN A MATHEMATICAL PROGRAMMING SYSTEM, I

by

J.A. Tomlin
Ketron Management Science,Inc., Mountain View, CA 94040

and

J.S. Welch
Ketron Management Science,Inc., Arlington, VA 22209

1. Introduction

For over thirty years Mathematical Programming Systems (MPS's) have been built arround sophisticated implementations of the simplex method, with its many variations, refinements and extensions. During almost this entire period the simplex method was unchallenged as *the* method of choice for solving linear programs, and by extension those other classes of problems, such as mixed integer and some kinds of nonlinear programs, which use sequences of linear programs (LP's). In the light of new developments in mathematical programming, this situation now requires re-examination.

The projective method, published by Karmarkar (1984), is an *interior point* method, which at first sight has nothing in common with the simplex method, proceeding as it does through the (relative) interior of the feasible region, rather than from vertex to vertex of the polytope defining that region. While claims have been made for the theoretical virtues of other methods (notably the ellipsoid method of Kachiyan(1979)), Karmarkar's work has drawn much attention because of widely publicised claims that his method is orders of magnitude *faster* than the simplex method and some of the commercial MPS's that employ it.

Much of the controversy remains to be resolved. It turns out that, viewed in the geometry in which Dantzig originally concieved the simplex method (see Dantzig(1963)), the two methods are not wholly unrelated (Stone and Tovey(1986)). Even more intriguingly, it has been shown by Gill et al (1985) that the projective method is equivalent to a special case of a barrier function method (see Fiacco and McCormick(1968)). This has lead to a much better understanding of the mathematical and computational proceedures involved, which is now being backed up by independant computational results. In particular, Gill et al (1985) obtained computational results showing the barrier method to be comparable in efficiency with the simplex method. Subequently, Adler et al (1986) presented computational results which indicate that an affine form of the projective method (see Vanderbei et al (1985)), operating on the dual LP can be faster than some simplex implementations by a factor of two or more on many problems.

Even though these computational results are very preliminary, they are impressive enough to warrant a realistic evaluation of interior point methodologies in the context of the large scale MPS's actually used to solve large scale real world problems. Such integration and testing is the topic of this paper.

2. Special Purpose Codes and MPS's

Development of new mathematical programming algorithms (usually for problems of particular structure) has frequently lead to the development of special pupose codes, usually written in FORTRAN or some other high level language. These are often used to test the algorithms and then "cleaned up" for use on practical problems. However, in most cases the facilities provided are rudimentary - the ability to input a model in standard form (such as MPS format), an optimizer, and solution output (perhaps also in MPS format) for the best solution obtained. It is rare to find facilities for revising and restarting models, parametrics or ranging, or any of the other "bells and whistles" provided by a large scale MPS (see Orchard-Hays (1968) for a description of MPS facilities and terminology). There are, of course, exceptions which prove the rule; notably the restart and model revision tools provided in MINOS (see Murtagh and Saunders (1985)) and the "X System" of Brown and Graves (1983).

A large scale MPS is a complicated software system which may be dominated not by algorithmic considerations, but by model management, sensitivity analysis, case study and reporting requirements. The end user usually does not know (or care) how the results were produced, but often requires voluminous reports and the ability to work from a large database. The resulting environment may then be something like that in Figure 1; a simplified schematic of Ketron Management Science's proprietary MPSIII system.

The data path on the right of Figure 1 uses time-honored MPS card-image input, which is CONVERTed to an intermediate packed form on the PROBFILE. This is followed by SETUP, which selects appropriate right hand side, objective and bound information for the specified case and creates a packed "workfile" or "SETUP problem". In MPSIII (as in all descendants of MPS/360) this is an out-of-core file. (See also Benichou et al.(1977)).

The data path on the left of Figure 1 uses the DATAFORM model management language to hold the database, build models, save solution cases, create reports, perform case studies, etc. Here models are saved in tree-structured form on the random-access ACTFILE, which contains *all* data pertinant to them, rather than on a PROBFILE (though conversion is possible between the two forms). The Program Control Language (PCL) verb READY performs analagously with SETUP in producing a workfile from the ACTFILE.

The workfile produced by these data processing steps is the model representation to be operated on by the algorithmic modules. These include the PRIMAL or VARIFORM algorithm (with the option of using GUB), DUAL and numerous Ranging and Parametric algorithms. Solution reporting, some forms of matrix editing, reoptimization from an old basis and many other functions must also be done at the workfile level.

MPSIII (and some other systems) may carry the above process a step further and repack the workfile to give an even more compact representation using the concept of "super-sparsity" (see Kalan(1971) or Greenberg(1978)). This concept makes use of the fact that the number of "unique values" even in a sparse matrix may be relatively small, and in particular, many of them may be +1 or -1. This may be taken advantage of by using a "pool" of unique values, which is referenced by the individual nonzeros and by avoiding multiplications by unit coefficients. Thus the standard workfile represents vectors (columns) as in Figure 2(a), while WHIZARD, our high-speed in-core optimizer uses a representation as Figure 2(b). If WHIZARD is used for rapid optimization, the solution information (which variables are basic and which at bound) must be returned to the model in standard workfile format, where we can then INVERT and recompute the solution in the required form for use by other modules.

It should be clear that such MPS's posses a great deal of power and flexibility which their developers and users are loath to lose or be forced to recreate. The question of when and if some algorithm other than the revised simplex method (around which such systems were designed) should be incorporated is thus a serious one. Beale and Tomlin (1970) elucidated three criteria for incorporating new algorithms:

(a) They are reasonably easy to implement.

(b) They are reasonably easy to use.

(c) *They enable a significant class of problems to be solved more easily in an MPS framework.*

Good examples satisfying these criteria have been GUB (Generalized Upper Bounding) and Branch and Bound methods. Given a super-sparse capability of the WHIZARD type, Tomlin and Welch (1984, 1985) showed that special primal simplex algorithms for pure and generalized networks were also excellent examples. On the other hand, attempts to incorporate decomposition as an integral part of MPS's (as opposed to user implementation - see Ho and Loute (1983)) have been less successful.

When we come to examine the newer interior point methods in this light, we find that criterion (b) is easily dealt with - the new algorithm is simply called in place of the old for primal optimization. Criterion (a) is the topic of most of the rest of this paper. Criterion (c) is of course still a matter of major controversy, but we have preliminary computational results which indicate positive results for at least one significant class of problems.

3. MPS Data Structures

We have already described some of the data structures used for matrix representation at the algorithmic level in MPSIII. We should also consider the related data structures used in the real work of the revised simplex method. These are the structures of the "transformations" (sometimes known as "etas") used to update vectors. The basis (or its inverse) is factorized into a product of elementary transformations:

$$B^{-1} = E_k....E_2E_1 \tag{3.1}$$

and is used to compute:

$$\{\alpha_1, \alpha_2, ..., \alpha_w\} = B^{-1}\{a_{j_1}, a_{j_2}, ..., a_{j_w}\} \tag{3.2}$$

and

$$\begin{Bmatrix} \pi^T \\ \gamma_1^T \\ \gamma_2^T \\ \vdots \end{Bmatrix} = \begin{Bmatrix} c^T \\ e_{p_1}^T \\ e_{p_2}^T \\ \vdots \end{Bmatrix} B^{-1} \tag{3.3}$$

Note that usually several vectors are transformed at once (see Orchard-Hays (1968) and Forrest and Tomlin (1972)). The nonunit column (or row) of the elementary transformations may be stored analagously with the way in which matrix columns are stored for the sparse and super-sparse structures.

Finally we indicate the way in which memory is allocated in the standard MPSIII SETUP mode and in WHIZARD in Figure 3. The WHIZARD structure is inherantly more flexible, and is the one we have used for other special algorithm implementations. Note that we have shown a larger partition for the WHIZARD structure, since the WHIZIN conversion module seeks out as large a work space as possible. The size, manageability and availability of this space (normally used for transformations) is important for interior point methods since they use a number of arrays of the *column* dimension of the LP. Allocation of such arrays in the SETUP mode would be very difficult.

4. Choice of Method(s)

Since the original appearance of Karmarkar's work there have been numerous variations and extensions, and a choice (or choices) for implementation must be made. We chose the Newton barrier method (see Gill et al (1986)) for our initial implementation for the following reasons:

(a) Familiarity, due to our collaboration on the cited reference.

(b) The fact that Karmarkar's projective method and the affine method of Vanderbei et al (1985) may be extracted as special cases of this method.

(c) The ability to handle problems in standard form, with bounds and ranges, with reasonable efficiency.

(d) The fact that it works with a *primal feasible* solution (if there is one). This allows for early feasible termination.

We shall not recapitulate the details of the algorithm (see Gill et al (1986)), merely give an outline which emphasizes the critical sub-algorithms so that we see how they may affect the MPS. The problem is stated as:

$$\min_{x} \sum_{j} (c_j x_j - \mu \ln x_j)$$

subject to:

$$Ax = b$$
$$0 \leq x_j \leq U_j$$

where $\mu \longrightarrow 0$.

To outline the algorithm let us define:

$$D = diag\{x_j\},$$
$$d = c - A^T \pi,$$
$$r = Dd - \mu e.$$

Then the steps of an iteration are:

(1) If μ and $\|r\|$ are sufficiently small - STOP.

(2) If "appropriate" reduce μ and recompute r.

(3) Solve a least squares problem:

$$\min_{\delta\pi} \|r - DA^T \delta\pi\|. \tag{4.1}$$

(4) Update the "pi values" and "reduced costs":

$$\pi \leftarrow \pi + \delta\pi, \quad d \leftarrow d - A^T \delta\pi,$$

and compute the search direction p as:

$$r = Dd - \mu e, \quad p = -(1/\mu)Dr.$$

(5) Calculate the steplength α.

(6) Update $x \leftarrow x + \alpha p$. GO TO (1).

The most critical step in each iteration is the gradient projection, that is solving the least squares problem (4.1). As in Gill et al.(1986) we use a preconditioned conjugate gradient method (the LSQR method of Paige and Saunders (1982)). This involves the Cholesky factorization of (an often approximate) normal equation matrix

$$PLL^T P^T = \bar{A}\bar{D}^2\bar{A}^T := AD^2A^T,$$

where the permutation matrix P is chosen so that the lower triangular factor L is sparse. We then solve the preconditioned problem

$$\min_{y} \|r - DA^T PL^{-T} y\| \tag{4.2}$$

and recover $\delta\pi = PL^{-T}y$.

The essential steps in LSQR are calculations of the form

$$u \leftarrow u + DA^T(PL^{-T}v), \tag{4.3}$$

and

$$v \leftarrow v + L^{-1}P^T(ADu). \tag{4.4}$$

5. Some Implementation Considerations

The WHIZARD environment turns out to be quite convenient for implementation of the Newton barrier method in the (assembly language) MPSIII system. A first advantage is that all of the Presolve/Postsolve facilities of WHIZARD are automatically available to reduce and simplify the model, in the same way as they are for WHIZNET (see Tomlin and Welch (1983,1985)). The existing WHIZIN module also handles memory management and allocation effectively. In particular most of the arrays automatically assigned when WHIZIN reconfigures memory for the simplex method have immediate analogues or obvious uses in the barrier context:

(1) The π vector is used in both algorithms.

(2) The β region usually used for the values of the basic variables holds the right hand side.

(3) The "basis headings" array or "H-region" is available for saving the row permutation P and its inverse.

(4) The 7 row-length work regions are available to hold the "artificial column" for phase I, $\delta\pi$, and the work vectors for LSQR.

The large transformation block may be segmented to store:

(5) The column-size vectors x, c, d, r.

(6) The preconditioner L.

There is one more large data structure, which allows access to A row-wise so that AD^2A^T may be computed. This is explained below.

An important decision had to be made on the extent to which we would choose data structures which accomodate existing subroutines and make implementation easier, particularly in the critical steps of computing and applying the preconditioner. A great deal of the simplex code *could* be "mined" for the barrier method. For example, note that the computational steps in (4.3) consist of updating v by L^{-T}, equivalent to a BTRAN of a pricing vector in the simplex method (3.3), and multiplication of DA^T by the resulting vector involves essentially the same work as a full (scaled) PRICE of the matrix. Similarly in (4.4) we see that forming ADu requires essentially the same work as performing a CHECK for a (scaled) solution u. The update of the permuted ADu by L^{-1} corresponds to the FTRAN of a vector in the simplex method (3.2). The computation of the Cholesky factors LL^Tthemselves could be regarded as a special symmetric application of the sparse factorization used in simplex INVERT routines.

It is a natural temptation to use the WHIZARD transformation data structures and modify INVERT, FTRAN, BTRAN etc. to exploit existing code. We decided *not* to do this for a number of reasons:

(a) We expect L to be not at all super-sparse, unlike the basis factors produced by INVERT for the simplex method, and indeed to be much denser altogether.

(b) The simplex FTRAN and BTRAN (see (3.2-3)) routines are designed with the efficient update of multiple vectors in mind. We do not wish to do that at this stage.

(c) Published algorithms are available (see George and Liu (1981)) which can be easily integrated, using simple data structures, and which predict storage requirements for L, thus enabling us to avoid exceeding space available.

We thus chose to adopt the data structures used by George and Liu (1981), including the use of the "compressed storage scheme", with the modification that the row index array is of full word length so that the indices multiplied by eight may be stored (to avoid shifts in inner loops). Pointers into the index and value arrays are also shifted appropriately.

6. Auxilliary Processing

There are several non-trivial proceedures which must be carried out before the algorithm can be even begun. In particular, the permutation P, the nonzero structure of L and a representation of A^T must be determined.

To determine P we require the non-zero structure of AA^T. We do this by processing the columns of A sequentially and building a list of index pairs (i_1, i_2) in each column, each pair in a full word. When all columns have been processed, or periodically if array space is exceeded, this list is sorted (using Shell sort). Duplicate pairs may then be purged in a single pass, and the data reconfigured into a form suitable for a minimum degree ordering, which gives P (see Liu (1985)). This is then followed by a symbolic factorization, derived from George and Liu (1981). As stated above, the data structure is modified for efficient application of the preconditioner L. Once the nonzero structure of L is available, the memory requirements for the algorithm can be computed, and if insufficient space is available the proceedure may be halted.

The remaining major data structure is required so that we may compute

$$LL^T = P^T \bar{A} D^2 \bar{A}^T P \tag{6.1}$$

where $\bar{A}$ and D are modified to exclude dense columns and those corresponding to tiny x_j. This requires either a much more expensive updating proceedure for the data structure L, if we access A only by column, or the ability to access the matrix row-wise. We chose the latter alternative. The data structure used to do this is as follows:

Since we already have a count of the number of nonzeros in each row from earlier processing it is easy to set up an array which contains a single word for each useful nonzero (i.e. those not eliminated by PRESOLVE) in the matrix. A row length array points to the first entry for each row (see Figure 4) in permutation order. Each entry in the ROWLNK array consists of a 3-byte column number for the element and a 1-byte offset which gives the offset of the nonzero in the super-sparse packed WHIZARD matrix. There is also a column-length COLADR array (also used for other purposes) which gives the address of each column in the matrix and a 1-byte offset which specifies where the non-unit elements begin relative to the top the column description. The offsets in the two arrays may thus be compared logically to see if a nonzero in the row being referenced is a unit entry or not, and treat it accordingly. Note that the ROWLNK array, once allocated, can be built in a single pass of the matrix, using only row-size work arrays. We should also point out that the 1-byte offsets limit this structure to columns represented by at most 256 bytes. However, this is no restriction, since such columns would contain over 120 nonzeros, and we do not wish to consider such columns when computing the preconditioner anyway.

7. Transition to a Basic Solution

No matter how rapidly an interior point algorithm may perform, it will not usually lead to a in instantly identifiable optimal *basic* solution. However in many, if not most applications a basic solution (or at least one with a minimal number of nonzero values) is very important. Reasons for this are:

(1) It is desireable in practice to keep the number of "active" activities small.

(2) Basic solutions are more "nearly integer" in many models, e.g. production-distribution models.

(3) It is easier to save and restore model status.

(4) All the standard postoptimality and sensitivity analysis procedures, including ranging and parametrics, assume a basic optimum.

(5) Use of nonlinear procedures requiring repeated solution of modified LP problems (e.g. MIP by branch and bound and SLP) seem to remain dependant on variants of the simplex method.

The idea of using some non-simplex method to perform part of the optimization work and then switching to the simplex method is by no means new. In fact a "BASIC" technique for purifying non-basic solutions has been a part of MPS's for many years (it appears in the early MPS/360 documentation, see also the MPSIII User's Manual). Among the uses that have been made of this proceedure are transitions from Dantzig-Wolfe decomposition (see Ho and Tomlin (1977)), from an Augmented Lagrangian method (see Beale, Hattersley and James (1985)) and projective/barrier methods (see Gill et al.(1986)). However, in the latter case the transition was carried out through an external interface, which is inefficient and clumsy. The clear remedy is to implement a BASIC procedure internally in WHIZARD to process the nonbasic solution. We realized this, but made an attempt to avoid it.

Our experiments with interior point methods had lead to the observation that there were invariably less than m solution values significantly away from zero (or their bounds). In other words, our problem is not an excess of candidates for basic status. It therefore seemed possible that a simple classification and pivoting scheme would suffice.

Let us assume some general "significance tolerance" ϵ and suppose we have obtained "optimal" primal and dual values x and π. The latter are used to calculate the reduced costs (d_j values) and we classify the columns and rows as in Table 1. The action to be tried for relevant pairs of categories, based on complementarity

Cols / Rows	$x_j > \epsilon$ $d_j < \epsilon$	$x_j < \epsilon$ $d_j < \epsilon$	$x_j < \epsilon$ $d_j > \epsilon$
Slack$< \epsilon$ $\lvert\pi_i\rvert > \epsilon$	PIVOT HERE	SECONDARY PIVOT	FIX TO ZERO
Slack$< \epsilon$ $\lvert\pi_i\rvert < \epsilon$	SECONDARY PIVOT	TERTIARY PIVOT	
Slack$> \epsilon$ $\lvert\pi_i\rvert < \epsilon$	BASIC SLACK		

Table 1

considerations, is given in the body of the table.

The primary attempt is to pivot columns with positive values on active constraints (with $\pi_i \neq 0$). Normally there would be an imbalance of these, which requires a secondary choice of either pivoting columns with positive x_j on rows with $\pi_i := 0$, or columns with a zero x_j on active constraints. There may then be some tertiary pivots necessary to use up the remaining rows which are not assigned basic slacks. If all went well, this scheme should lead to an optimum basic feasible solution.

In practice this naive scheme almost always failed. This seems to be because, even for very sparse and degenerate problems, there is significant linear dependance between the columns corresponding to positive x_j in the solutions produced by the interior point method. Attempts to pivot these columns into a basis therefore are doomed to failure, and we are in the same position as when a singular "basis" is provided by a user - columns must be dropped, and the starting solution is infeasible. The number of simplex iterations needed to recover feasibility and then optimality may be large.

Given the failure of the naive scheme, there was no alternative to implementing a version of the BASIC algorithm within WHIZARD itself. The steps of this algorithm are not widely known, so we briefly restate them here:

Suppose all variables are at tentative values x_j^* and that:

$$Ax^* = b, 0 \leq x_j^* \leq U_j$$

The normal simplex iteration, pivoting on row r, computes:

$$\beta_i \leftarrow \beta_i - \alpha_{ij}\theta_r \qquad (i \neq r)$$

when column j enters the basis. [illegible] assume that, as usual, if a vector has $d_j > 0$ it will be decreased (as if it was coming in from it's upper bound). In this case $-a_j$ will be FTRANned and used in CHUZR. If $d_j < 0$ it is to be increased in the usual way. The explicit calculations are:

Case (1) $d_j < 0$

If $\theta_r < (U_j - x_j^*)$ do a regular pivot. i.e. modify β by $\theta_r \alpha_j$. Set $\beta_r = \theta_r + x_j^*$.

If $\theta_r \geq (U_j - x_j^*)$ do a bound flip type move. i.e. modify β by $(U_j - x_j^*)\alpha_j$ and mark x_j as being at it's upper bound (of U_j).

Case(2) $d_j > 0$

If $\theta_r \geq x_j^*$ do a bound flip type move. i.e. modify β by $x_j^* \alpha_j$ and mark x_j as being at it's lower bound (of 0).

If $\theta_r < x_j^*$ do a regular pivot. i.e. modify β by $\theta_r \alpha_j$. Set $\beta_r = x_j^* - \theta_r$.

Depending on the starting nonbasic solution there is no guarantee that the result will be feasible or optimal, only that the sum of infeasibilities or objective will not be degraded. It is therefore necessary to call the simplex method to verify or achieve optimality before entering Postsolve.

8. Preliminary Computational Results

It is not our intention to give detailed or extensive computational experience in this paper (this will be done in a later publication). Some preliminry results are of interest, however. Table 2 gives the original and reduced (Presolved) dimensions of a subset of the problems in Gill et al.(1986). Note the significant reductions achieved for some models, particularly the highly degenerate NZFRI forestry model. If the large structurally degenerate part of the model is not removed in this class of modeis the behaviour of the simplex method can be badly affected, even using some anti-degeneracy procedure, whereas we see in Table 3 that the WHIZARD "fast primal" starting from a "crashed" basis easily outperforms the current interior point implementation. This is true of the other models also, with one exception.

Problem Name	Original Dimensions		Reduced Dimensions	
	Rows	Columns	Rows	Columns
AFIRO	28	32	28	32
SHARE2B	97	79	97	79
BRANDY	221	249	131	222
BANDM	306	472	244	322
DEGEN2	445	534	442	534
NZFRI	624	3521	415	2034

Table 2

The model DEGEN2 is also highly degenerate, but does not exhibit a significant structurally degenerate, removable component of the model. The simplex method requires a great many iterations for this class of models and the barrier method is able to win by a significant margin, even with quite a lengthy BASIC and simplex phase added on to achieve a basic optimum. We have therefore found at least one class of models for which interior methods show promise. More generally, we should point out that it seems that many of the models used elsewhere in comparisons to show interior methods to advantage seem to be highly degenerate. Furthermore, most comparisons have been made with MINOS, which is a FORTRAN code with no anti-degeneracy procedure and no Presolve to remove structural degeneracy. Our comparisons, on the other hand, are with a state-of-the-art assembly language simplex code, with both algorithms in the same environment.

Problem Name	WHIZARD FAST PRIMAL	WHIZARD BARRIER	BARRIER PLUS BASIC
AFIRO	0.06	0.018	0.020
SHARE2B	0.36	0.96	1.14
BRANDY	1.32	3.72	4.50
BANDM	1.86	5.28	7.02
DEGEN2	35.58	18.54	24.00
NZFRI	6.06	20.16	29.64

Table 3

9. Further Work

The implementation and results we have described represent only a beginning, with the need for a number of improvements recognized. The efficiency of the WHIZARD internal BASIC procedure can certainly be improved. Almost certainly, other improvements can be made to the interior point method itself. We have not yet tried the straightforward Vanderbei et al.(1985) algorithm or the approximate trajectory method used in Adler et al.(1986). Similarly there are possible improvements to be gained by using different sparse factorization techniques along the lines given by Gustavson et al.(1970). The possibility of updating the LL^T factors, as suggested by Karmarkar (1984) and Shanno (1985) has yet to be evaluated, as have dual algorithms (Osborne (1986)).

A pressing question for general use of interior methods in MPS's is whether restart procedures for modified models can be found which compete with variants of the simplex method. Until this question is answered there is no alternative to using simplex-based technology for such procedures as branch and bound, sequential LP, or even regular production use of many large-scale LP models.

10. Conclusion

Despite the dramatic differences in philosophy and approach, it seems that interior point LP methods can be implemented with reasonable efficiency, and even elegance, in production Mathematical Programming Systems based on the simplex method. What is not yet known is what effect this will have on the economical solution of which classes of large-scale LP models. We have tentatively identified one class, and certain other models with special structure (e.g. block angular and staircase) seem good candidates. We expect to report further on implementation and computational experience in subsequent papers.

Acknowledgements

The authors are indebted to T. A. Dehne and M. A. Saunders for a number of helpful suggestions and to C. J. Strauss for his assistance with computational experiments.

References

Adler, I., Resende, M. G. C. and Veiga, G. (1986). An Implementation of Karmarkar's Algorithm for Linear Programming, Manuscript, Dept. of IE/OR, Univerity of California, Berkeley, CA.

Beale, E. M. L. (1970). "Advanced Algorithmic Features for General Mathematical Programming Systems," in *Integer and Nonlinear Programming*, (J. Abadie, ed.), pp. 119-137, North Holland Publishing Company, Amsterdam.

Beale, E. M. L. and Tomlin, J. A. (1970). "Special Facilities in a General Mathematical Programming System For Non-convex Problems Using Ordered Sets of Variables", in *Proceedings of the Fifth International Conference on Operational Research* (J. Lawrence, ed.), pp. 447-454, Tavistock Publications, London.

Beale, E .M. L., Hattersley, R. and James, L. (1985). An augmented Lagrangian approach to linear programming. Paper presented to the 12th International Symposium on Mathematical Programming, Boston, MA., August 1985.

Benichou, M., Gauthier, J. M., Hentges, G. and Ribière, G. (1977). The efficient solution of large-scale linear programming problems — some algorithmic techniques and computational results, *Math. Prog.* **13**, pp. 280–322.

Brown, G. G. and Graves, G. W. (c.1983). *XS Mathematical Programming System*, perpetual working paper.

Dantzig, G. B. (1963). *Linear Programming and Extensions*, Princeton University Press, Princeton, New Jersey.

Fiacco, A. V. and McCormick, G. P. (1968). *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*, John Wiley and Sons, New York and Toronto.

Forrest, J. J. H. and Tomlin, J. A. (1972). Updating triangular factors of the basis to maintain sparsity in the product form simplex method. *Math. Prog.* **2**, pp. 263–278.

George, J. A. and Liu, J. W. (1981). *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ.

Gill, P. E., Murray, W. and Wright, M. H. (1981). *Practical Optimization*, Academic Press, London and New York.

Gill, P. E., Murray, W., Saunders, M. A., Tomlin, J. A. and Wright, M. H. (1985). On Projected Newton Barrier Methods for Linear Programming and an Equivalence to Karmarkar's Projective Method, *Math. Prog.* **36**, pp. 183–209.

Greenberg, H. J. (1978). "A tutorial on matricial packing", in *Design and Implementation of Optimization Software*, (H. J. Greenberg, ed.), pp. 109–142, Sijthoff and Noordhoff, The Netherlands.

Gustavson, F. G., Liniger, W. and Willoughby, R. (1970). Symbolic generation of an optimal Crout algorithm for sparse systems of linear equations. *Journal of the ACM* **17**, pp. 87–109.

Ho, J. K. and Loute, E. (1983). Computational experience with advanced implementation of decomposition algorithms for linear programming. *Math. Prog.* **27**, pp. 283–290.

Ho, J. K. and Tomlin, J. A. (1977). "A hybrid approach to multi-stage linear programs". Technical report SOL 77-27, Department of Operations Research, Stanford University, Stanford ,CA.

Kalan, J. E. (1971). "Aspects of large-scale in-core linear programming", in *Proceedings of the ACM Annual Conference, Chicago, Ill.*, pp. 304–313, ACM, New York.

Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming, *Combinatorica* **4** pp. 373–395.

Khachiyan, L. G. (1979). A polynomial algorithm in linear programming, *Doklady Akademiia Nauk SSSR Novaia Seriia* **244**, pp. 1093–1096. [English translation in *Soviet Mathematics Doklady* 20, (1979), pp. 191–194.]

Liu, J. W. H. (1985). Modification of the minimum-degree algorithm by multiple elimination, *ACM Transactions on Mathematical Software* **11**, pp. 141–153.

Murtagh, B. A. and Saunders, M. A. (1983). *MINOS 5.0* user's guide, Report SOL 83-20, Department of Operations Research, Stanford University, California.

M.R. Osborne (1986) "Dual Barrier Functions with Superfast Rates of Convergence for the Linear Programming Problem", manuscript, Dept. of Statistics, Australian National University, Canberra.

Wm. Orchard-Hays (1968). *Advanced Linear Programming Computing Techniques*, McGraw-Hill, New York.

Paige, C. C. and Saunders, M. A. (1982). *LSQR*: An algorithm for sparse linear equations and sparse least-squares, *ACM Transactions on Mathematical Software* **8**, pp. 43–71.

Shanno, D. F. (1986). Computing Karmarkar Projections Quickly. Working paper 85-10, Grad School of Admin, University of California, Davis, CA.

Stone, R. E. and Tovey, C. A. (1986). Karmarkar's algorithm as a generalization of simplex. Paper presented at the TIMS/ORSA 21st Joint National Meeting, Los Angeles, CA, April 1986.

Tomlin, J. A. and Welch, J. S. (1983). Formal optimization of some reduced linear programming problems, *Math. Prog.* **27**, pp. 232–240.

Tomlin, J. A. and Welch, J. S. (1984). Solving generalized network models in a general purpose mathematical programming system. Paper presented at the ORSA/TIMS 18th Joint National Meeting, Dallas, TX, November 1984.

Tomlin, J. A. and Welch, J. S. (1985). Integration of a Primal Simplex Algorithm with a Large Scale Mathematical Programming System, *ACM Trans. on Math. Softw.* **11**, pp. 1–11.

Vanderbei, R. J., Meketon, M. S. and Freedman, B. A. (1985). A modification of Karmarkar's linear programming algorithm, Manuscript, AT&T Bell Laboratories, Holmdel, New Jersey.

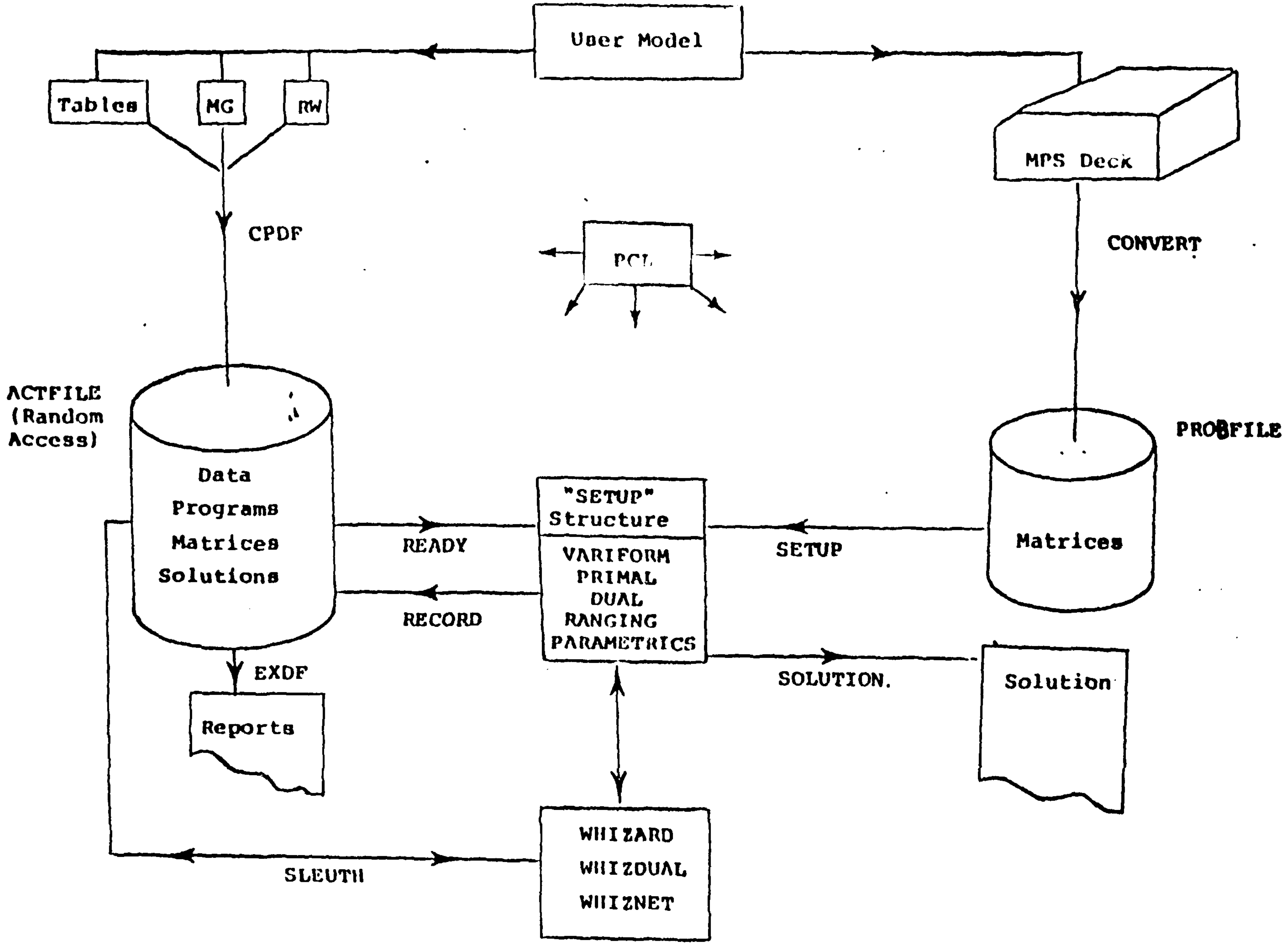


Figure 1.

MPSIII Structure

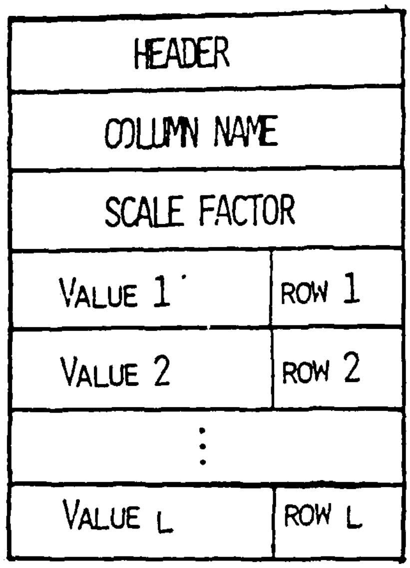


FIGURE 2(A)

WORKFILE FORMAT

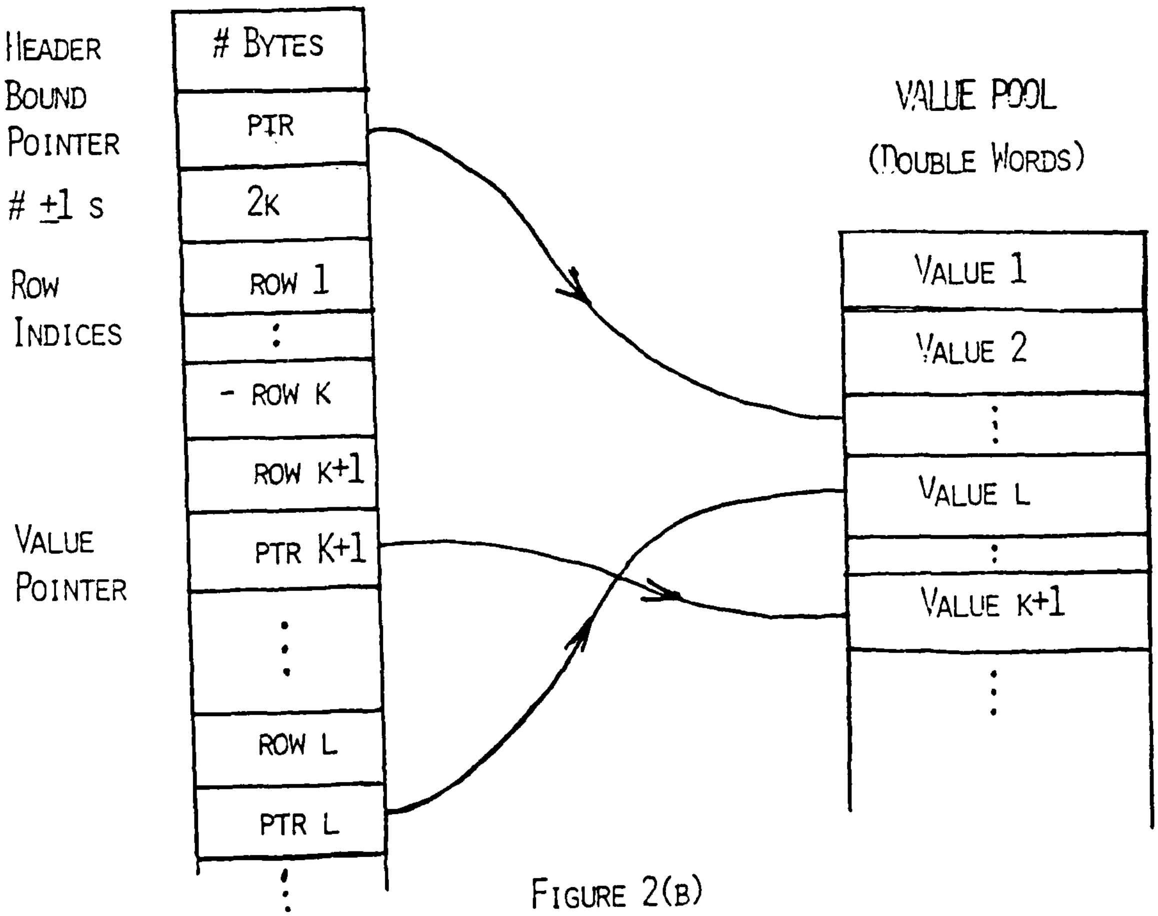


FIGURE 2(B)

WHIZARD FORMAT

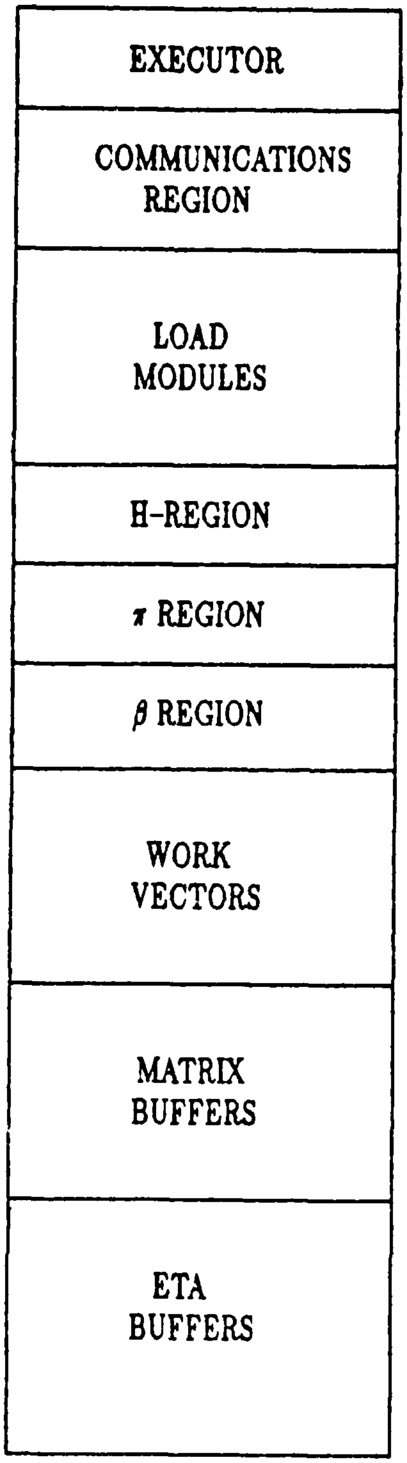


Figure 3(a)

SETUP Structure

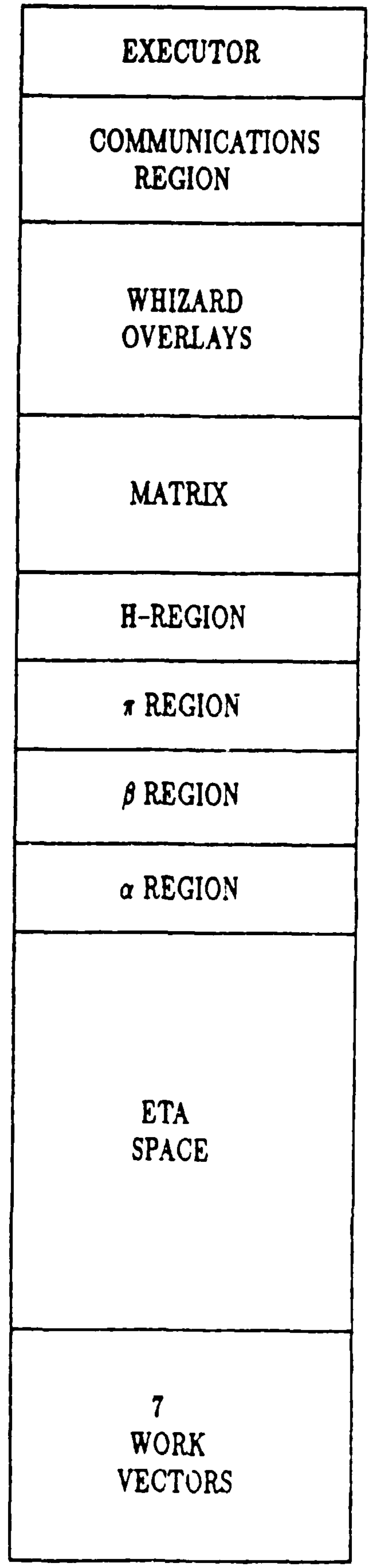


Figure 3(b)

WHIZARD Structure

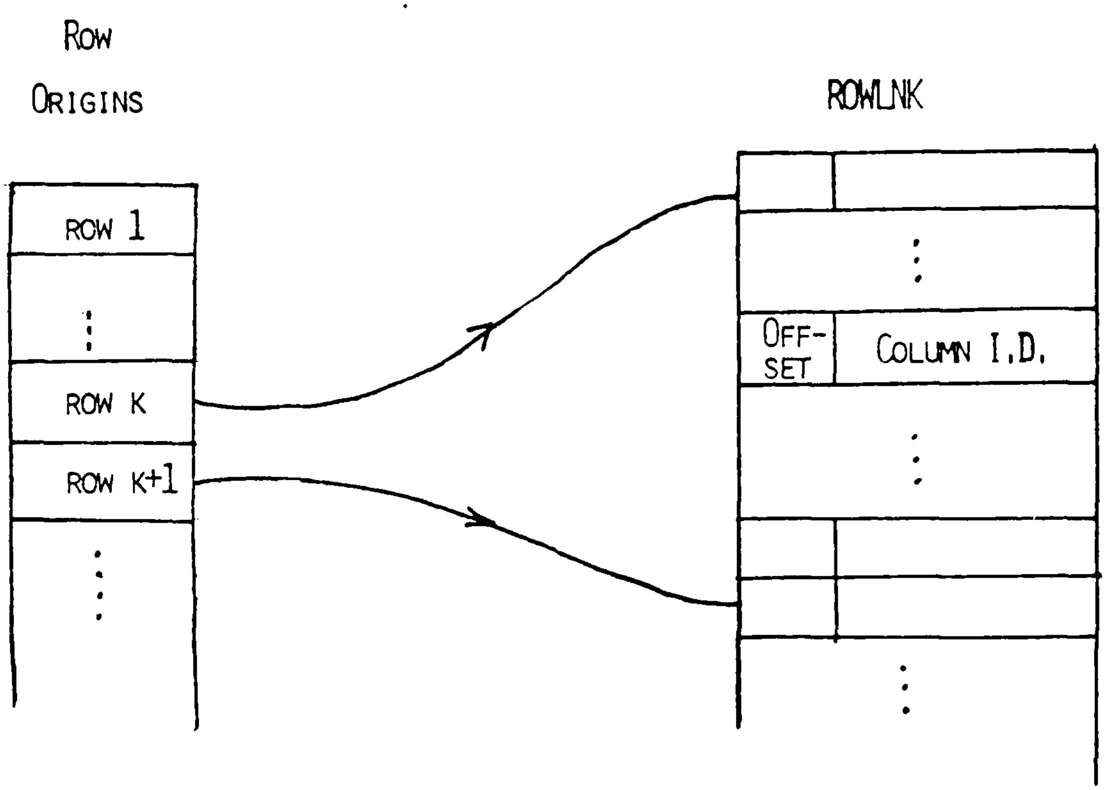


FIGURE 4.

ROW LINKS

END

4-87

DTIC